

# fgets() and gets\_s()

---

Robert C. Seacord, Software Engineering Institute [vita<sup>1</sup>]

Copyright © 2005, 2008 Pearson Education, Inc.

2005-09-27; Updated 2008-10-06

L4 / D/P, L<sup>2</sup>

The `gets()` function is a common source of buffer overflow vulnerabilities and should never be used. The `fgets()` and `gets_s()` functions each offer a more secure solution.

## Development Context

Reading strings from standard input

## Technology Context

C, UNIX, Win32

## Attacks

Attacker executes arbitrary code on machine with permissions of compromised process.

## Risk

The `gets()` function is a common source of buffer overflow vulnerabilities and should never be used. Programs running with elevated privileges, including programs that are outward facing, can be used for privilege escalation or to launch a remote shell.

## Description

If there was ever a hard and fast rule in secure programming in C and C++ it is this: Never use `gets()`. The `gets()` function reads a line from standard input into a buffer until a terminating newline or EOF is found. No check for buffer overrun is performed. The following quote is from the man page for the function:

Never use `gets()`. Because it is impossible to tell without knowing the data in advance how many characters `gets()` will read, and because `gets()` will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security.

There are two alternative functions that can be used: `fgets()` and `gets_s()`. Figure 1 shows how all three functions are used.

The `fgets()` function is defined in C99 [ISO/IEC 99] and has similar behavior to `gets()`. The `fgets()` function accepts two additional arguments: the number of characters to read and an input stream. By specifying `stdin` as the stream, `fgets()` can be used to simulate the behavior of `gets()`, as shown in lines 6-10 of Figure 1. Unlike `gets()`, the `fgets()` function retains the newline character, meaning that the function cannot be used as a direct replacement for `gets()`.

When using `fgets()` it is possible to read a partial line. It is possible, however, to determine when the user input is truncated because the input buffer will not contain a newline character. The `fgets()` function reads at most one less than the number of characters specified from the stream into an array. No additional characters are read after a newline character or after end-of-file. A null character is written immediately after the last character read into the array. The C99 standard does not define how `fgets()` behaves if the number of characters to read is specified as zero or if the pointer to the character array to be written to is a `NULL`.

---

1. [http://buildsecurityin.us-cert.gov/bsi/about\\_us/authors/274-BSI.html](http://buildsecurityin.us-cert.gov/bsi/about_us/authors/274-BSI.html) (Seacord, Robert C.)

The `gets_s()` function is defined by ISO/IEC TR 24731 [ISO/IEC 05] to provide a compatible version of `gets()` that is less prone to buffer overflow. This function is closer to a direct replacement for the `gets()` function than `fgets()` in that it reads only from the stream pointed to by `stdin`. The `gets_s()` function, however, accepts an additional argument of `rsize_t` that specifies the maximum number of characters to input. An error condition occurs if this argument is equal to zero or greater than `RSIZE_MAX`<sup>13</sup> or if the pointer to the destination character array is null. If an error condition occurs, no input is performed and the character array is not modified. Otherwise, the function reads at most one less than the number of characters specified and a null character is written immediately after the last character read into the array. Lines 11-15 of Figure 1 show how `gets_s()` can be used in a program.

**Figure 1. Use of `gets()` vs. `fgets()` vs. `gets_s()`**

```
1. #define BUFFSIZE 8
2. int _tmain(int argc, _TCHAR* argv[]){
3.     char buff[BUFFSIZE];
4.     // insecure use of gets()
5.     gets(buff);
6.     printf("gets: %s.\n", buff);
7.     if (fgets(buff, BUFFSIZE, stdin) == NULL) {
8.         printf("read error.\n");
9.         abort();
10.    }
11.    printf("fgets: %s.\n", buff);
12.    if (gets_s(buff, BUFFSIZE) == NULL) {
13.        printf("invalid input.\n");
14.        abort();
15.    }
16.    printf("gets_s: %s.\n", buff);
17.    return 0;
18. }
```

The `gets_s()` function returns a pointer to the character array if successful. A null pointer is returned if the function arguments are invalid, an end-of-file is encountered and no characters have been read into the array, or if a read error occurs during the operation.

The `gets_s()` function only succeeds if it reads a complete line (that is, it reads a newline character). If a complete line cannot be read, the function returns `NULL`, sets the buffer to the null string, and clears the input stream to the next newline character.

The `fgets()` and `gets_s()` functions can still result in buffer overflows if the specified number of characters to input exceeds the length of the destination buffer.

## References

- |              |  |
|--------------|--|
| [ISO/IEC 99] | ISO/IEC. <i>ISO/IEC 9899 Second edition 1999-12-01 Programming languages — C</i> . International Organization for Standardization, 1999.                   |
| [ISO/IEC 05] | ISO/IEC. <i>ISO/IEC TR 24731 Extensions to the C library -- Part 1: Bounds-checking interfaces</i> . International Organization for Standardization, 2005. |

---

13. The `RSIZE_MAX` is used to limit the size of objects passed to functions that have parameters of type `rsize_t`. Extremely large object sizes are frequently an indication that an object's size was calculated incorrectly. For example, negative numbers appear as very large positive numbers when converted to an unsigned type like `size_t`. Also, some implementations do not support objects as large as the maximum value that can be represented by type `size_t`. As a result, it is sometimes beneficial to restrict the range of object sizes to detect potential vulnerabilities.

## Pearson Education, Inc. Copyright

---

This material is excerpted from *Secure Coding in C and C++*, by Robert C. Seacord, copyright © 2006 by Pearson Education, Inc., published as a CERT® book in the SEI Series in Software Engineering. All rights reserved. It is reprinted with permission and may not be further reproduced or distributed without the prior written consent of Pearson Education, Inc.